



1. Descrição geral

A avaliação da cadeira de aplicações distribuídas está dividida em quatro projetos. O Projeto 3 não tem ligação com os dois anteriores.

O objetivo geral do presente projeto será concretizar um serviço Web para gerir um sistema simplificado de classificação de músicas de utilizadores. A implementação vai utilizar o estilo arquitetural REST [1] e uma base de dados relacional acessível pela linguagem SQL. Para este efeito, serão utilizados a *framework* de desenvolvimento Web *Flask* [2] e o motor de base de dados SQL *sqlite* [3] no servidor. O programa cliente utilizará o módulo *requests* [4] para implementar a interação cliente/servidor baseada em HTTP.

2. Esquema da base de dados

A definição da base de dados assenta nos conceitos envolvidos: utilizador, música, artista, avaliação e lista de músicas. Cada lista de músicas criada por um utilizador contém uma ou mais músicas e cada música está associada a um artista (podendo várias músicas estarem associadas ao mesmo artista). Cada conceito corresponde a uma tabela de acordo com a Figura 1, onde também se ilustram as várias relações.

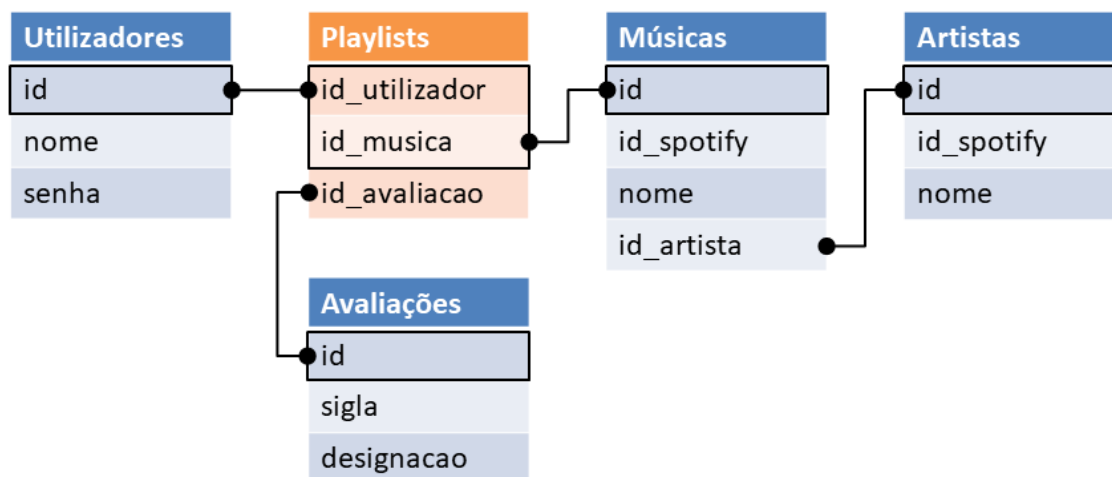


Figura 1 - Esquema da base de dados.

Para criar a base de dados, poderá ser utilizado o código SQL apresentado da Listagem 1. A primeira linha desta listagem serve para que o *sqlite* possa suportar chaves estrangeiras. Por omissão, na instalação de alguns sistemas operativos, essa opção está desabilitada. Repare também na opção “ON DELETE CASCADE”, a qual serve para indicar que a remoção de um elemento numa tabela pai, causará a remoção de todas as linhas que apontavam para este elemento numa tabela filha.

```

PRAGMA foreign_keys = ON;

CREATE TABLE utilizadores (
    id                INTEGER PRIMARY KEY,
    nome              TEXT,
    senha             TEXT
);

CREATE TABLE musicas (
    id                INTEGER PRIMARY KEY,
    id_spotify         TEXT,
    nome              TEXT,
    id_artista         INTEGER,
    FOREIGN KEY(id_artista) REFERENCES artistas(id) ON DELETE CASCADE
);

CREATE TABLE artistas (
    id                INTEGER PRIMARY KEY,
    id_spotify         TEXT,
    nome              TEXT
);

CREATE TABLE avaliacoes (
    id                INTEGER PRIMARY KEY,
    sigla             TEXT,
    designacao        TEXT
);

CREATE TABLE playlists (
    id_user            INTEGER,
    id_musica          INTEGER,
    id_avaliacao       INTEGER,
    PRIMARY KEY (id_user, id_musica),
    FOREIGN KEY(id_user) REFERENCES utilizadores(id) ON DELETE CASCADE,
    FOREIGN KEY(id_musica) REFERENCES musicas(id) ON DELETE CASCADE,
    FOREIGN KEY(id_avaliacao) REFERENCES avaliacoes(id) ON DELETE CASCADE
);

```

Listagem 1 SQL para criar as tabelas para o projeto.

Os dados inseridos na tabela `avaliacoes` são os que se seguem e poderão ser inseridos pelo seguinte código SQL.

```

INSERT INTO avaliacoes (id, sigla, designacao) VALUES
    (1, "M", "Mediocre"),
    (2, "m", "Mau"),
    (3, "S", "Suficiente"),
    (4, "B", "Boa"),
    (5, "MB", "Muito Boa")
;

```

Listagem 2 Código para inserir dados na tabela `avaliacoes`.

A aplicação pretendida deverá contemplar uma rotina de inicialização que verifica se a base de dados já existe. Caso esta não exista, ela deverá ser criada e inicializada com o código apresentado (i.e., a criação das tabelas e a inserção dos registos na tabela `avaliacoes`).

3. O programa cliente

O programa cliente aceita interactivamente quatro operações e os seus parâmetros (de forma semelhante à leitura do *stdin* como nos projetos anteriores), e comunica com o servidor para que este processe as operações e armazene a informação numa base de dados. A Tabela 1 mostra detalhadamente as operações que o cliente deverá suportar.

Tabela 1 - Lista de operações que o cliente aceita e parâmetros correspondentes.

Operação	Parâmetros	Observações
CREATE	UTILIZADOR <nome> <senha> ou ARTISTA <id_spotify> ou MUSICA <id_spotify> ou <id_user> <id_musica> <avaliacao>	Se ao criar uma nova música, o artista desta música ainda não tiver sido inserido na base de dados, o servidor deverá inseri-lo. A última variante serve para o utilizador avaliar uma música, onde $avaliacao = M m S B MB$
READ ou DELETE	UTILIZADOR <id_user> ou ARTISTA <id_artista> ou MUSICA <id_musica> ou ALL < UTILIZADORES ARTISTAS MUSICAS > ou ALL MUSICAS_A <id_artista> ou ALL MUSICAS_U <id_user> ou ALL MUSICAS <avaliacao>	As quatro últimas variantes permitem obter: <ul style="list-style-type: none"> • todos os utilizadores, músicas ou artistas • todas as músicas avaliadas de um artista; • todas as músicas avaliadas por um utilizador; • todas as músicas avaliadas com uma dada avaliação (M, m, S, B, MB);
UPDATE	MUSICA <id_musica> <avaliacao> <id_user> ou UTILIZADOR <id_user> <password>	

Convém lembrar que os identificadores (i.e., as chaves primárias) dos elementos nas tabelas de utilizadores, músicas, artistas e avaliações são números inteiros crescentes.

O cliente comunicará com o servidor através de mensagens em HTTP e usará uma representação utilizando JSON [5]. Para este efeito, os alunos utilizarão o módulo *requests* [4]. As mensagens terão de respeitar a API REST definida pelo serviço Web de criação de listas de músicas.

4. O serviço Web

O serviço Web será implementado com recurso à *framework* Flask [2] e a API REST será disponibilizada através de três URLs de base:

1. /utilizadores
Para operações relativas aos utilizadores.
2. /artistas
Para operações relativas aos artistas.
3. /musicas
Para operações relativas às músicas.

É muito importante que os alunos planeiem a API REST antes de iniciarem a implementação. Sugere-se que façam uma tabela onde definam a correspondência entre as operações suportadas, o método do HTTP, as URLs dos recursos, os parâmetros das operações, e as possíveis respostas HTTP com que o serviço responderá ao cliente.

Quando o serviço recebe uma mensagem de um cliente, a operação deverá ser implementada sobre a base de dados e a resposta será preparada segundo os padrões REST utilizando JSON para transmitir a representação dos recursos ou o conteúdo de outras mensagens.

Para o acesso à base de dados, os alunos devem procurar na documentação sobre o Flask a forma de fazer com que a ligação à base de dados exista de forma automática sempre que a aplicação recebe um pedido.

5. Integração com serviço REST público da Internet

De forma a obter mais informações sobre as músicas e os artistas, os alunos devem recorrer à informação pública disponível sobre os mesmos. No âmbito da disciplina, iremos recorrer à API REST disponível no *Spotify* [6]. A informação disponível no *Spotify* deve completar a informação disponível no servidor a construir, i.e., cada vez que a informação relativa a uma música ou artista for requerida ao servidor Flask, esta deve ser complementada com a informação vinda do *Spotify*. No entanto, as informações vindas do *Spotify* não serão guardadas localmente (excepto o campo `nome`). As informações serão apenas apresentadas ao utilizador, requerendo um novo pedido de cada vez que se queira listar a informação sobre as músicas e os artistas. Neste projeto **não** é permitido utilizar bibliotecas cliente (e.g., *Spotipy*) para a interação com a API Web do Spotify.

Note que o *Spotify* utiliza uma codificação base-62 nos seus identificadores, pelo que eles serão armazenados como texto ao invés de números inteiros na base de dados do projeto. Dois exemplos de identificadores do *Spotify* para um artista e uma música são:

- **Artista:** Xutos & Pontapés ([1lQnDEcvFAWaUjbyZiHKih](https://open.spotify.com/artist/1lQnDEcvFAWaUjbyZiHKih))
- **Música:** Contentores ([4YK4SXRtVreEWvJNOdg3wW](https://open.spotify.com/track/4YK4SXRtVreEWvJNOdg3wW))

Autenticação e autorização à API REST do Spotify:

Para ter acesso à API REST do Spotify é preciso seguir alguns passos de autenticação e autorização:

1. Entrar em <https://developer.spotify.com/dashboard/>
2. Fazer o login ou criar uma nova conta.
3. Registar uma aplicação. Para isso basta definir o nome da aplicação e descrever brevemente o objetivo da aplicação.
4. Copiar o Client ID e o Client Secret
5. Ler atentamente a [documentação da API Web do Spotify](#).
6. Executar alguns testes simples na [Spotify Web API Console](#).
7. Após preencher os campos do pedido, é preciso gerar um token OAuth para estar autorizado a executar o comando.
8. Executar alguns testes simples com o comando `curl`, como no exemplo a seguir. Pode-se substituir o campo `nome_artista` pelo nome do artista que pretende buscar na plataforma, assim como deve-se substituir o campo `meu_OAuthToken` pelo token obtido no passo anterior:

```
curl -X "GET"
"https://api.spotify.com/v1/search?q=nome_artista&type=artist" -H
"Accept: application/json" -H "Content-Type: application/json" -H
"Authorization: Bearer meu_OAuthToken"
```
9. Para o Projeto 3, é suficiente copiar manualmente o token obtido no Passo 7 para uma variável no código do programa servidor. Este token possui uma duração limitada, o implica que este passo possa ter de ser refeito algumas vezes durante o desenvolvimento do projeto.

6. Tratamento de erros

Sempre que a operação não possa ser executada ou que algum erro inesperado ocorra, o servidor deverá responder ao cliente com uma resposta HTTP incluindo uma descrição detalhada do problema segundo o formato apresentado nas aulas TP sobre a representação JSON (i.e., `application/api-problem+json` [7]). O cliente apresentará a informação da descrição detalhada na consola.

7. Entrega

A entrega do Projeto 3 consiste em colocar todos os ficheiros do projeto numa diretoria cujo nome deve seguir exatamente o padrão `grupoXX` (onde `XX` é o número do grupo). Juntamente com os ficheiros, deverá ser enviado um ficheiro de texto `README.txt` (é em TXT, não é PDF, RTF, DOC, DOCX, etc.) onde os alunos podem relatar a informação que acharem pertinente sobre a sua implementação do projeto (por exemplo, limitações). A diretoria será incluída num ficheiro ZIP cujo nome deve seguir exatamente o padrão `grupoXX.zip` (novamente `XX` é o número do grupo). Esse ficheiro será submetido num recurso a disponibilizar para o efeito na página de AD no moodle da FCUL. O não cumprimento destas regras podem anular a avaliação do trabalho.

O prazo de entrega do Projeto 3 é **domingo, 1º de Maio de 2022, às 23:59 (WEST)**.

8. Plágio

Não é permitido aos alunos dos grupos partilharem código com soluções, ainda que parciais, a nenhuma parte do projeto com alunos de outros grupos (nem através do Fórum da disciplina, nem por qualquer outro meio). Além disso, todos os códigos serão testados por um sistema verificador de plágio. Caso alguma irregularidade seja encontrada, os projetos de todos os alunos envolvidos serão anulados e o caso poderá ser reportado aos órgãos responsáveis na Ciências@ULisboa.

9. Referências

- [1] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- [2] <https://flask.palletsprojects.com>
- [3] <https://www.sqlite.org/>
- [4] <http://docs.python-requests.org/en/master/>
- [5] <http://www.json.org/>
- [6] <https://developer.spotify.com/documentation/web-api/reference/>
- [7] <https://datatracker.ietf.org/doc/html/draft-nottingham-http-problem-02>